マルチプロセッサLSIに適したオンチップ・ネットワーク

——ダイレクト・データフロー・インターコネクト

デバイスの記事

東原朋成

プロセッサ・インターフェースとして一般的なバス・システムは，マルチプロセッサ・システムではシステム性能のボトルネックになりやすい．ここでは米国Tensilica社が同社のXtensa LX向けに提供しているTIEポートというダイレクト・データフロー・インターコネクトを例に，マルチプロセッサLSIに適したオンチップ・ネットワークについて考える．（編集部）

米国Tensilica社のXtensaは，システムLSI設計のためのコンフィギャラブルで拡張性のあるプロセッサ・コアです．先ごろ発表されたXtensa LXという新しいアーキテクチャでは，

- XPRES：コンパイラ技術
- FLIX：プログラムのコード・サイズを最適化しつつ，VLIW（very long instruction word）技術を取り入れた命令拡張
- TIEポート/TIEキュー（以下，TIEポートと呼ぶ）：システム開発者が定義できるプロセッサ・インターフェース

などの技術が取り入れられました．本稿では，これらの新しい技術の中から，TIEポートというこれまでのプロセッサにはないインターフェースについて解説します．

1．バス・インターフェースの問題

マイクロプロセッサのインターフェースというと，バス構造（アドレス，データ，コントロール）をとるのが一般的です．これは，米国Intel社の8080，米国Motorola社（現Freescale Semiconductor社）の6800といった8ビットCPUの時代からあまり変わっていません．これらのバスに直結できる各種周辺LSIがCPUメーカのみならず多くの会社で開発されて流通し，デファクト・スタンダードになっています．

システムLSIにおいても，例えば英国ARM社はAMBA（Advanced Microcontroller Bus Architecture）というバスを開発し，公開しています．このAMBAバスに直結するIP（intellectual property）コアは，数多く開発されて流

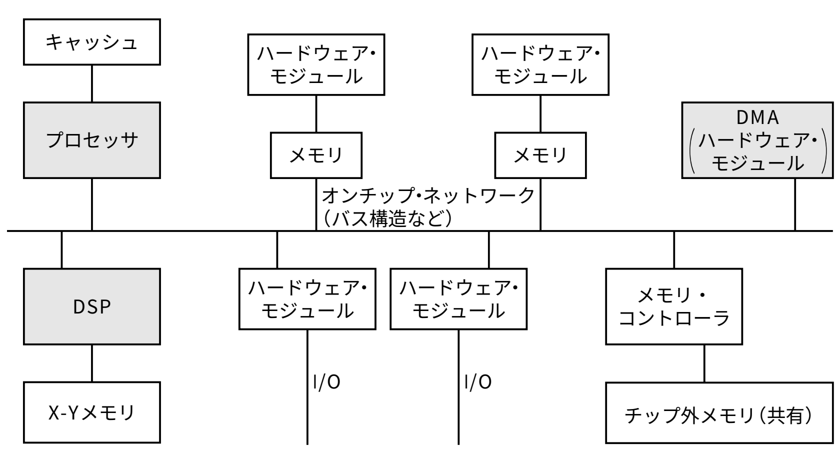


図1 一般的なシステムLSIの構造

オンチップ・ネットワークによってプロセッサやメモリ，ハードウェア・モジュールなどを接続する．バス構造が一般的だが，クロスバ・スイッチやメッシュ構造など，複雑なオンチップ・ネットワークを採用したシステムLSIも設計されている．

通し，デファクト・スタンダードになっています．ここでもプロセッサ・バス（インターフェース）という概念が踏襲されています．

典型的なシステムLSIの構成を**図1**に示します．この図で示すオンチップ・ネットワーク部分が，おのおののコンポーネント（プロセッサやメモリ，ハードウェア・モジュールなど）を接続する媒体です．バス構造が一般的ですが，クロスバ・スイッチやメッシュ構造など，複雑なオンチップ・ネットワークを採用したシステムLSIも設計されています．設計エンジニアが独自に開発することもできますが，デファクト・スタンダードのバスをそのまま利用するというケースが多いようです．

● マルチマスタ時のオーバヘッド

図1では，バス・マスタと呼ばれるバス・トランザクションを開始するモジュールが複数存在します．プロセッサやDMA（direct memory access）などがバス・マスタ・モジュールの代表です．

このような場合，一般にアクセスを調停するアービタが必要になります．複数のバス・マスタから同時にアクセス要求があった場合，アクセス権をとれなかったモジュールは待たされることになります（**図2（a）**）．つまり，アクセス遅延（オーバヘッド）が生じます．また，SDRAMなどへのアクセスの場合，メモリ・アクセスの遅延も加わり，アクセス遅延はますます大きくなります．

複数のマスタがアクセスするということは，必然的にデ

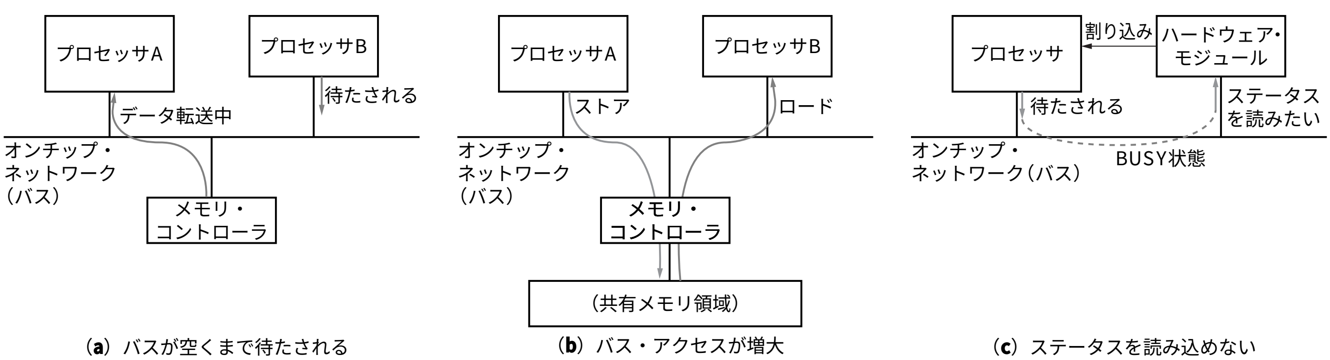


図2　バス・システムで起こる問題の例

（a）におけるプロセッサBの要求は，プロセッサAのデータ転送が終了するまで待たされる．プロセッサBが次にバスを獲得できる保証はないので，プロセッサBは長時間待たされるかもしれない．すると，プロセッサBの実質演算能力が低下することになる．（b）のように，共有メモリを通してデータのやり取りを行うと，バス・アクセスが増大する．（c）のように，ハードウェア・モジュールが割り込みを発生すると，プロセッサはハードウェア・モジュールのステータスを読み込んで次のコマンドをセットしようとする．プロセッサはシステム・バスを介してアクセスするが，バスの状態によっては待たされる可能性がある．すると，ハード・リアルタイムの（時間に対して非常に厳しい）システムでは，要求される時間制約を満足できずシステム・フェイルになる．

コラム　バス性能がシステム性能を決める

プロセッサの演算能力からくる限界によって，マルチメディア・アプリケーションなどではマルチプロセッサ（マルチマスタ）システムがより重要になると予想されます．例えば，HDTV（high definition television）レベルのMPEGビデオ・デコードのように高い性能が要求される場合，ASIC設計上の制約などからクロック周波数をひたすら上げていくことそのものに無理があります．

筆者が過去に手がけたMPEG（ビデオ，オーディオ）を扱うシステムLSI（ASIC）では，少なくとも二つのプロセッサと32チャネルのDMA（各チャネルは同時にバスを要求できる）を搭載していました．MPEGビデオなどは，要求される演算能力が数GOPS（giga operations per second）にも達します．プロセッサのほかに，いわゆるアクセラレータとしていくつものハードウェア・モジュールを搭載したので，多チャネルDMAエンジンが必要だったのです．

筆者が設計したLSIにおいて性能を左右したのは，演算能力ではなく実はバスでした．バスの最大データ転送量は，シェアしているメモリ（例えば外部DDR SDRAM）のデータ転送量より大きくする必要があります．バス・アクセスができるだけ平均化するよう，ピーク時のデータ転送要求がバスの最大データ転送速度を超えないようにソフトウェアを最適化しました．しかし，ある条件下で要求量が許容量を超えることを確認しました．また，別の条件下では許容量以下だったのですが，アクセス遅延が予想以上に大きくなり，ハード・リアルタイムのクリティカル時間を超えてしまい，不つごうが発生する可能性があることを発見しました．

ソフトウェアでバス・アクセス・スケジュールを調整して問題を解決できなかったため，問題を生じたハードウェア・モジュールのバッファ容量を大きくするというハードウェアの変更で対処することを検討しましたが，非常に高コストになることが判明しました．プロジェクト・スケジュールなど，ほかの要素を検討した結果，性能を落とすということでこの問題を解決せざるをえなかったことを，今でも残念に思っています．

ータの転送量が増大します．例えば，二つのプロセッサを用いてマルチメディア処理を行う場合，データをプロセッサ間で共有することが考えられます．共有メモリはバス上に置かれ，複数のマスタによる共有メモリへの読み書きが繰り返されます(**図2(b)**)．これが，マルチプロセッサ(マルチマスタ)システムにおけるデータ転送のオーバヘッドとなります(左掲のコラム「バス性能がシステム性能を決める」を参照)．

処理ユニット(プロセッサ，ハードウェア・モジュール)を100％動作させるには，データ，命令(プログラム，コントロール)が途切れることなく各ユニットに供給できなくてはなりません．データ・バスの容量(最大データ転送量)や最大遅延を考慮して，バッファリングやスケジューリングを行うわけですが，それらの見積もりは簡単ではありません．しかも，設計途上の仕様変更(おもに機能追加)が頻繁に行われるため，ますます見積もりは困難になっています(下掲のコラム「バスの性能を上げる」を参照)．

● バス幅の増加やバスの多重化における設計制約

システムLSIにおいても，8080，8086，68000などのプロセッサと同様にバス(例えばAMBA)が主流になっているのはなぜでしょうか．システムLSIではネットワークがチップ内に組み込まれます．ピン数の制約はあまり受けないはずです．汎用CPU(マイクロプロセッサ)では1,024ビット幅の外部バスを定義することは，ピン数の点から非現実的です．しかし，チップ内のCPUコアであれば十分可能だといえます．

コラム　バスの性能を上げる

クロスバ・スイッチや階層構造といった複雑なネットワークを設計することで，バス(ネットワーク)の性能を引き上げようとしたことがあります．

筆者の手がけたディジタル・テレビ向けMPEGビデオ・デコーダLSIでは，2本のデータ専用バス，および制御専用バスを設けました．そして，遅延に対してクリティカルな制御信号をバースト主体のデータ・アクセスと切り離し，しかもバースト・アクセスをメモリに対するデータの方向で二つに分離(いわゆるスプリット・トランザクション)するといったことまで取り入れ，かなり複雑なネットワークを形成しました．

しかし，筆者がこういった複雑な構造をとることができたのは，プロセッサに至るまですべて自分たちで設計していたからです．

例えば，ARM社が2003年6月に発表したAMBA 3.0では，1,024ビットまでのデータ幅をサポートするバス仕様「AXI(Advanced eXtensibles Interface)」が新しく追加されました．AMBA 3.0ではプロセッサ・インターフェースとシステム・ネットワークを区別していますが(AXIはプロセッサ・インターフェースの仕様)，この点が従来のAMBAバスと比較した場合の大きな相違点です．

Tensilica社のXtensa LXでは，最大128ビット幅で完全2重のプロセッサ・インターフェースを定義しています．例えば200MHzのクロックで理論上の最大データ・レートは6.4Gバイト/sです．キャッシュ以外のメモリにはDMAによってデータとプログラムを独立して転送できるので，理論上の最大データ・レートは12.8Gバイト/sともいえます．

しかし，システム全体で考えると，これだけのデータ・レートを実現することはやさしくはありません．共有メモリからくる制限，システム・バスのクロックの制限(バスの負荷が大きくなるとクロックを下げる必要がある．またはバッファを挿入するためにアクセス遅延が増える)，バス幅を増やすことに伴うレイアウト上の制限といった数々の問題があります．また，バス・アクセスの調停(アービタ)に伴うオーバヘッドも見過ごすことはできません．

2．ダイレクト・データフロー・インターコネクト

従来のバス・システムにおける問題を解決する一つの手法として，Xtensa LXでは「TIEポート」と呼ぶ新しいインターコネクトをシステム設計者が定義できるしくみを導入しました．TIEポートはダイレクト・データフロー・イ

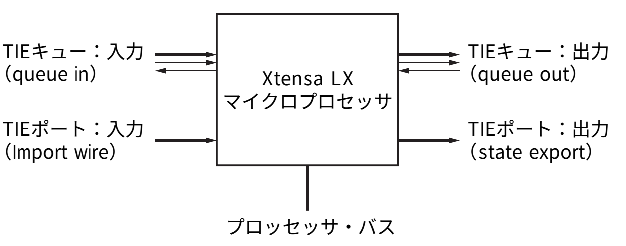


図3　TIEポート(ダイレクト・データフロー・インターコネクト)
TIEポートは，システム設計者が定義できるプロセッサI/Oインターフェースである．データ幅や入出力の方向，タイプ(Queueタイプなど)を設定できる．また，それぞれのタイプのポートを複数定義することができる．TIEポートを扱う命令語もシステム設計者が定義できる．Queueタイプのインターフェースでは，Queueを制御する出力(push：queue out，pop：queue in)，およびQueueの状態(full: queue out，empty：queue in)が自動的に定義される．

ンターコネクトとも呼べる技術です．プロセッサ間のデータをやり取りする際に，従来のプロセッサ・インターフェースの機構（バス）をバイパスします．

図3のように4タイプのTIEポートがあります．各ポートは，データ幅に上限はあるものの自由に定義できます．例えば9ビット幅のTIEポートも定義できます．1バイトを最小単位とする従来のプロセッサ・インターフェースとはかなり様子が異なります．

Queueタイプ（queue in，queue out）は，FIFOなどのようなキュー構造のバッファとインターフェースをとります．キューの状態を知らせる入力（full，empty）やキュー・アクセスを知らせる出力（push，pop）の信号が自動的に生成されます．

state exportは，ユーザが定義したXtensa LXプロセッサのステート・レジスタの内容をプロセッサ外部に出力したものです．これを使って，例えばハードウェア・モジュールやほかのXtensa LXプロセッサを直接制御できます．

import wireは，プロセッサ内部に外部の状態を取り込

コラム FLIXの概要

Xtensa LXのもう一つの重要な技術であるFLIXについて簡単に解説します．詳細はTensilica社のホームページ（http://www.tensilica.com/）などを参照してください．

プロセッサの演算能力を高める方法にはアーキテクチャ・レベル（ISA：Instruction Set Architectureレベルと呼ばれることもある）とマイクロアーキテクチャ・レベル（おおざっぱには，実際の設計時に検討するレベル）の方法があります．これらの境界はきれいに分けられるものではなく，非常にあいまいです．例えば学術的に考えるとパイプライン，キャッシュなどはマイクロアーキテクチャ・レベルになるのですが，キャッシュをアクセスする命令（とくに初期化のための命令）は通常必要になるので，ちょっと矛盾しそうです．プロセッサによってはキャッシュ・アクセス命令などを特別扱い（互換性を保証しないなど）するケースもありますが，あいまいであることに違いはありません．

さてプロセッサの処理能力を高めるには，

1）1クロックでできるだけたくさんのことを実行させる
2）1クロックでできるだけたくさんのデータを処理する
3）ロスを少なくする
4）クロック周波数を上げる

が大きな要素になります．**4）**は実際の設計によるので，マイクロアーキテクチャの領域と言えるでしょうか．クロック周波数を上げる方法は非常に有効ですが，クロック周波数を上げ続けることは消費電力などの要素まで絡めて考えると非常に難く，ASIC設計のフローではおのずと限界があります．

1）については，例えばいわゆるCISC型アーキテクチャのプロセッサには複雑な演算を実行する命令があったので，それを1クロックで実行できればRISC型アーキテクチャのように簡単な命令語ベースのプロセッサと比較して効率が良いといえます．また，命令に要するメモリも節約できます．RISCの概念では二つの値の差をとって差の絶対値を求めるという演算は2命令で実行することになります．しかし，アプリケーションでこの演算を多用することがわかっている場合，差の絶対値を求める演算命令があれば，処理効率を高められるうえ，メモリも節約できます．クロック周波数の問題も出てきますが（差を求めるだけの演算のほうがやはり速くできる），1クロックでできるだけたくさんのことを実行させるには有効な方法です．

これとは別に，スーパスカラと呼ばれる，1クロックで複数の命令語を同時に実行する方法があります．これは，RISC型プロセッサが処理効率を上げるときの方法として大きく発達しました．2，4，8個のデコーダ，演算ユニットを持たせ，同時に複数命令を実行させようという発想ですが，分岐命令などが存在するため，つねに並列実行できるわけではなく，しかも処理がかなり大掛かりになるため，組み込み用途のプロセッサではあまり見かけません．スーパスカラはマイクロアーキテクチャに属するといえるでしょう．

VLIWはこの発想をアーキテクチャ・レベルで実現したといえるかもしれません．通常のプロセッサの命令語はたかだか32ビットですが，VLIWでは128ビットや256ビットといった非常に長い命令語を使います．命令語をいくつかのスロットに区切り，スロットごとに演算を割り当てます．もっとも各スロットにつねに有効な演算を割り当てることは難しく，しばしばNOP（何もしない）演算を割り当てることになります．

2）については，例えばベクトル量を同時に扱えれば，行列演算を多用するアプリケーションなどで処理能力が格段に向上します．スーパコンピュータで花開いたアーキテクチャですが，Intel社のMMXをはじめとしたマルチメディア・アプリケーション用途でこの手法はかなり一般的になっています．

3）については，ハイエンドのプロセッサでは分岐予測といった複雑な技術を実現しています．分岐予測はマイクロアーキテクチャに属する要素なのですが，コンパイラは分岐ロスや分岐予測についてもよく認識している必要があります．

おもしろいもので，コンパイラはアーキテクチャを逸脱することはできません．しかし，マイクロアーキテクチャをよく理解していないと性能を引き出すことはできません．事実，コンパイラをバージョンアップすると，処理能力が向上することがあります．

現在のところ，本プロセッサ・アーキテクチャはスーパスカラ構成をとっていません．回路規模が非常に大きくなるためです．また，分岐予測のような複雑なマイクロアーキテクチャ構造もとっていません．パイプラインは至ってシンプルです．本プロセッサではシステム・エンジニアがアプリケーションに最適な命令語を定義できます．上述した差の絶対値を求めるという命令やさらに複雑な演算の定義も可能です．ここで注意していただきたいのは，「複雑な

む入り口を提供します．例えばハードウェア・モジュールがプロセッサに割り込みをかけたとき，割り込みルーチンではimport wireを介してハードウェアの状態を直接読み込めます．

TIEポートを用いると，多少の制限はあるものの，システムに応じて必要なタイプやデータ幅をかなり自由に設計できます．**図4**に二つのキュー入力を定義した例を示します．各タイプのポートを複数定義することはもちろん可能です．従来のプロセッサ・インターフェースの場合，（コンフィギャラブルなプロセッサ・コアでさえも）自由度は限られています．

● TIEポートのアクセスにはユーザ定義命令を使う

TIEポートは，プロセッサ間，およびプロセッサとハードウェア・モジュールの間を直接接続します．

TIEポートを直接アクセスする命令はシステム設計者が定義します．通常，プロセッサ間を直接アクセスする命令はないので，これまでとはずいぶん趣きが異なるといえる

演算」イコール「回路が複雑で遅い」，ということがかならずしも成立しないことです．アプリケーションに最適な命令語を定義することで，処理能力を何十倍にも高めることが可能です．また，データ・タイプを新たに定義することもできるので，ベクトル・タイプのデータを定義すると，データの同時処理（SIMD）も実現できます．

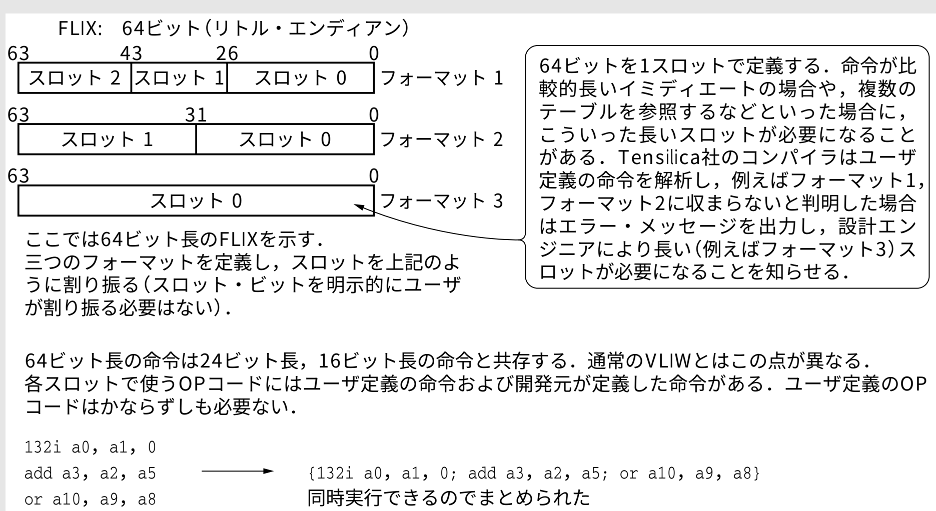
さて，本プロセッサは，FLIXと呼ぶ新たに発展したVLIWの考えかたに属する技術を採用しています．Xtensaで定義できるのは16ビット，24ビットの命令語ですが，本プロセッサではこれらに加えて64ビット（もしくは32ビット）の命令語も定義できます．64ビットの命令語のフォーマットは不定であり，ユーザがほぼ自由に定義できます．複数のスロットに分けることもできれば，一つのスロットとして扱うこともできます．1スロットのフォーマットである場合，より長いイミディエートや複数のテーブル変数を扱うといった命令を定義できるようになります．もちろん複数フォーマットを指定することも可能です．

FLIXと通常のVLIWを比較した場合の大きな相違は，VLIWでしばしば発生するNOP命令挿入をFLIXではほとんどなくすことができることです．VLIWは命令語長が固定ですが，FLIXアーキテクチャのプロセッサは16，24，64ビットという3種類のフォーマットを持つため，NOPの挿入を極力回避できます（**図A-1**）．

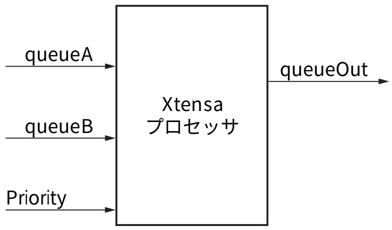
ユーザは明示的に各スロットを定義できます．いくつのスロットに分けるかだけを指定すれば，ツールが自動的にスロットを定義するので，ユーザの手を煩わせることはありません．各スロットはあらかじめ開発元が用意した命令を使用することもできます．実際にコンパイラは，**図A-1**に示したような三つの24ビット命令を一つの64ビット命令にまとめることもします．24ビット×3は72ビットであり，64ビットには収まらないように見えます．しかし，同じ演算を行うだけで命令語のフォーマットはかならずしも元の24ビットのフォーマットと同じである必要がないため，このようなことが可能になります．

本プロセッサといっしょに提供されるコンパイラ（C，C++）はFLIXをサポートしています．そのため，ユーザはFLIXを意識せずにプログラムを書くことができます．コンパイラはプログラムを解析し，性能とコード・サイズを最適化したバイナリ・コードを出力します．FLIXオプションをONにすることで，同じソース・コードでも数十倍の性能向上が図れます．FLIXの技術とユーザ独自の実行命令を定義すること，データ・タイプを定義することなどで，本プロセッサは従来のプロセッサ・コアの何十倍もの演算性能を得ています．

このような演算能力の高いプロセッサに途切れることなくデータを供給する一つの手法がTIEポートなのです．



図A-1
FLIX
三つのコア命令を，FLIX技術を用いて64ビットの1命令として実行するようにした．このようにXtensa LXのコンパイラはソース・コードを解析して，実行とプログラムのコード・サイズの最適化を行う．この例では三つのコア命令を一つにまとめるため，実行性能が上がる．また，三つの24ビット命令を64ビットにまとめるので，メモリも節約できる．

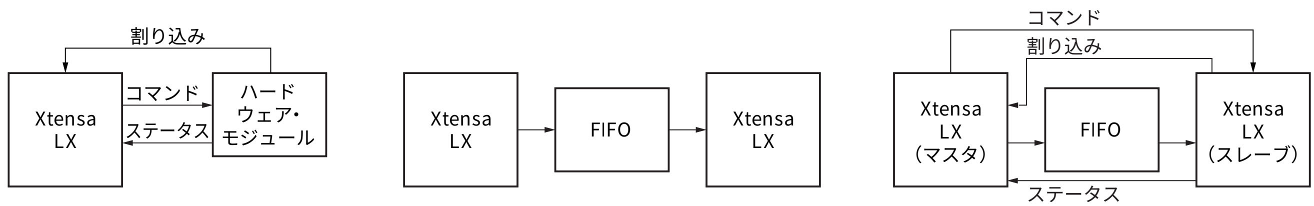



```
operation queueMUL{}{in queueA,in queueB,in Priority,out queueOut,out valid}
{
 wire[31:0]MUL=queueA*queueB;
 queueOut=Priority?MUL[31:16]:MUL[15:0];
}

queueA,queueB:16ビット,TIE queue in
queueOut:16ビット,TIE queue out
Priority:1ビット import wire（入力）
```

図4　TIEポートを用いたユーザが定義した命令語の例

2入力のキューを読み込んで積をとる．Priorityが“H”の場合は演算結果の上位16ビットを，そうでない場合は下位16ビットを出力する．queueMULは命令語としてプログラムで使用できる．



（**a**）ハードウェア・モジュールの直接制御　（**b**）FIFOを使う方法　（**c**）FIFOと直接制御の両方を使う方法

図5　TIEポートを用いた事例

（a）のXtensa LXはハードウェアを直接制御できる．ハードウェアが割り込みをアサートした時点でステータスを直接読み込め，ハードウェア制御の遅延を最小にできる．（b）では，FIFOを使ってXtensa LX間でデータをやり取りする．フラグを使わず，FIFOの状態（full，empty）によって同期をとることができる．（c）はより高度な使いかたで，FIFOを通してデータをやり取りするだけでなく，例えばスレーブ側のXtensa LXのタスク終了をマスタ側に割り込み（スレーブのTIEポートがマスタの割り込み端子につながっている）で伝えると同時に，ステータスをマスタ側に返す．マスタはスレーブの動作をコマンドを使って制御できる．

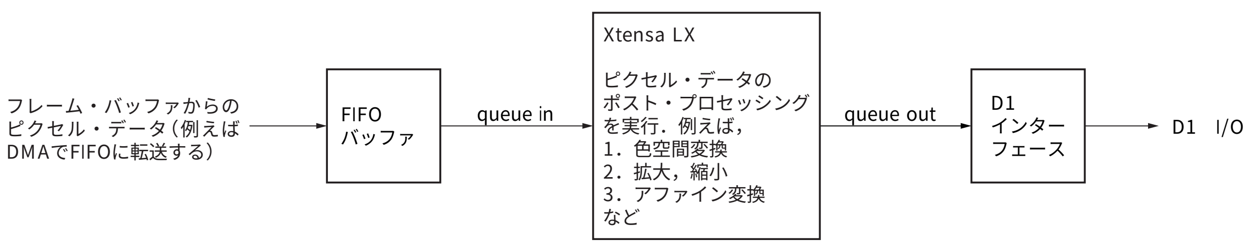


図6　TIEポートを用いたシステム設計の一例

この例では，DMAなどを使ってピクセル・データをFIFOバッファに取り込み，そのピクセル・データをソフトウェアで変換した後，直接，D1インターフェース（ハードウェア）に出力している．バス方式をとる場合，変換したピクセル・データをいったんフレーム・バッファなどに書き込み，そのデータをDMAなどでD1インターフェースに転送する必要がある．したがって，システム・バスの負荷が増大するとともに，バスにおける遅延に対処するため，多大なバッファをD1インターフェースなどが持つ必要がある．また，変換したピクセルを書き込むという作業はバスの有効利用の観点から非効率である．

かもしれません．ポートを直接アクセスするので，外部メモリのような遅延の問題はありません．

プロセッサ間で例えばロード命令を実行するとき，そのロードがプロセッサ・インターフェースをすぐにアクセスするのかどうかは不明（ある条件下では自明）であり，不確定要因の一つになります．TIEポートでは，基本的にアービタは関与しません（ただし，システム設計によってはアービタを使うようなやりかたももちろん可能）．

TIEポートから直接プログラムをフェッチして実行するという機能は現在のところはありませんが，理論的にはデータとして扱うことができるものはすべてTIEポートを通じてやり取りできます．プログラムもそれをデータとみなしてTIEポートを通して取り込み，プログラム・メモリにストアすることで，ほかのプロセッサ（例えばスレーブ・プロセッサ）に渡すことが可能です．

● システム性能が向上

TIEポートの使用例を**図5**と**図6**に示します．**図5**のようにプロセッサ間や，プロセッサとハードウェア・モジュールの間をTIEポートを使って接続すると，高いデータ・レートを得ることができます．インターコネクトの負荷が共有バスと比べて軽くなれば，高いクロック・レートで動作します．また，必要に応じてデータ幅を増やすこともできます．バスを経由する必要性がなくなるので，不確定要素を極力排除できます．かつシステム・バスのアクセス負荷も軽減できます．

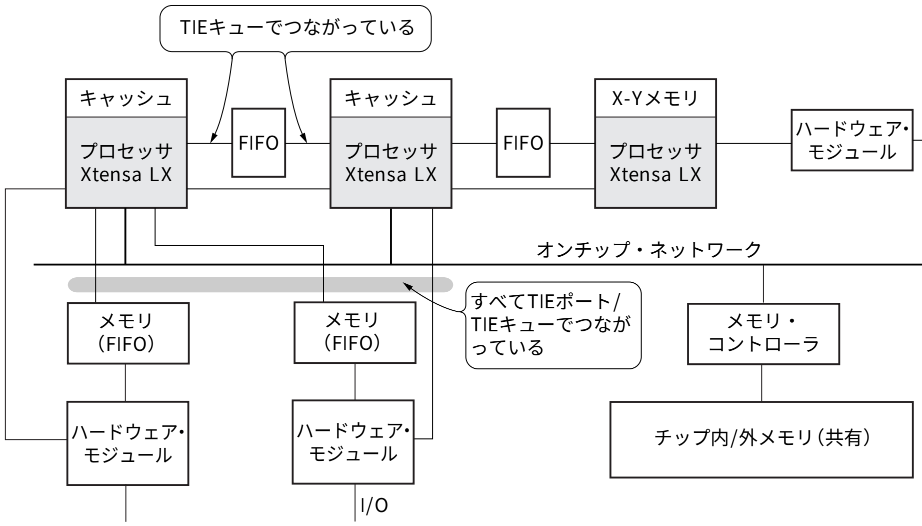


図7
TIEポートを活用したシステムLSIの構成例
システム・バスとTIEポートは共存する．

TIEポートでは，プロセッサからハードウェア・モジュールを直接アクセスできます．例えば，割り込み発生時にプロセッサはハードウェア・モジュールの状態をTIEポートを通して直接知り，直接コマンドを与えられます．バス・アクセスなどによる不確定な遅延も排除できるので，ハード・リアルタイム・システムにおけるクリティカル時間違反によるシステム・フェイルといった現象を回避することが可能になります．

● マルチマスタの動作をシンプルに実現

TIEポートのような直接的な（密結合と言ってよいかもしれない）インターフェースは，使いかたが特殊と感じるかもしれません．マルチプロセッサのシステムLSIは珍しくありませんが，二つ以上のマスタがデータを共有したり，一つのハードウェア・モジュールが二つのマスタによって制御されるといったシステムはあまりみかけません．制御が複雑になりすぎて，設計や検証が非常に困難になる場合が多いからです．

ここで，二つのプロセッサがデータを共有するケースを考えてみます．小さな作業メモリをオンチップ上に持つとします．そしてこのメモリにはプロセッサ1が書き込み，それをプロセッサ2が読み出します．同期にはセマフォ・フラグを使います．このシステムでは通常，バス上においてお互いのプロセッサがセマフォを監視しながらデータの受け渡しを行うので，必然的にバス・アクセスが増大します．ところが，この作業メモリが二つのプロセッサでのみ共有されるのならば，システム・バスを介する必要はありません．そこでデータのやりとりをローカルで実行できるように，階層構造を持ったネットワークを設計することがあります．しかし，アドレス・デコードと比べると複雑な論理を設計する必要があります．

一方，共有データをTIEポートを介してやり取りすれば複雑なネットワーク構造をとる必要がありません．Queueタイプの TIEポートは同期機構を有するのでセマフォを介する同期も不要になります．

図7にTIEポートを使ったシステムLSIの構成例を示します．**図1**と比較してみてください．**図7**に示すように，システム・バスとTIEポートは共存します．システム設計者は自分たちが解決しようとするシステムの要求に応じて使い分けるものです．

TIEポートは，マルチプロセッサ・システムのためだけのものではありません．キャッシュ・メモリやX-Yメモリなどのプロセッサのローカル・メモリ・インターフェースに，メモリ以外のモジュールを接続することは困難です．ところがアプリケーションによっては，CAM（content addressable memory；連想メモリ）やテーブル・ルックアップといった異なるメモリにアクセスしたい，と考えることがあります．例えばMPEGなどで採用されているハフマン・コードの復号は，テーブル参照ができれば非常に効率良く実行できます．ソフトウェアでは込み入った配列の参照という形で実行することになります．TIEポートを使えば，テーブル参照も至って簡単に実現できます．CAM構造のメモリなどが簡単に接続できるようになるからです．これもTIEポートの柔軟性を示す一つの好例です．

とうはら・ともなり
米国Tensilica，Inc.